This is a draft article that outlines a line of research into a promising technology, which forms one of the components of our comprehensive approach to developing inter-agent communication systems.

Teaching LLMs to Speak in Pseudocode for Efficient Communication

Can large language models invent their own compressed "language" without losing meaning? In this article, we explore a novel approach: using reinforcement learning to train an LLM to translate natural language into a compact meta-language pseudocode. The goal is to pack the same information into far fewer tokens – improving speed, cost, and memory – while preserving full semantic meaning (including critical details like numbers). Unlike standard data compression (e.g. ZIP files), the output here is a structured sequence of normal tokens (a kind of pseudocode) that any LLM can later read and reconstruct without special tools. We'll discuss why such semantic compression is valuable, how an RL-based training pipeline might achieve it, examples of the pseudocode in action, potential applications, and how this idea relates to existing research.

Background: Why Compress Language Semantically?

In today's AI systems, token efficiency is increasingly important. Large Language Models (LLMs) often face context window limits – they can only process a few thousand tokens at a time, yet real tasks (legal documents, codebases, medical records) often far exceed these limits . Even when context size grows to 100k tokens, feeding huge prompts or histories remains expensive in terms of computation, latency, and API costs. Every token in a prompt or response costs time and money to process, so reducing tokens while keeping essential information can yield big gains. As researchers from Microsoft note, prompts today can "exceed tens of thousands of tokens", leading to high inference cost and latency . Compressing prompts or context without losing fidelity can accelerate inference and reduce cost .

Natural language is also highly redundant. Claude Shannon observed in 1951 that everyday language contains a lot of predictable or repetitive content . This means there's room to shorten messages dramatically. Humans summarize information, but summaries often drop specifics or nuance. What we want instead is a lossless (or near-lossless) semantic compression: a way to say the same thing in fewer words. If an AI truly "understands" the data, it should be able to express it succinctly without omitting details. In fact, recent advances suggest a deep connection between understanding and compression. "If you understand something, you can express it in very short form," says AI researcher Ming Li – his team showed that large models can compress text far more than traditional algorithms by guessing/predicting content intelligently . In other words, compression is a test of comprehension.

This motivates training an LLM to generate a compact pseudocode that encodes all the meaning of the original input. Such a semantically dense representation could be transmitted or stored and later expanded back to full text by another model. The benefits would be broad: more content squeezed into a prompt (important for Retrieval-Augmented Generation and long documents), lower storage and bandwidth needs, and faster inference since the model reads fewer tokens. Researchers have already started probing this idea. For example, Gilbert et al. (2023) prompted GPT-4 to "produce compressed responses that still preserve rich semantic information" and found it could often condense text or code and later reconstruct it with high fidelity . In one case, GPT-4 was able to compress source code into a shorter description and recover the original functionality almost perfectly . These early studies treat the LLM as its own compression tool via prompting. The novel idea here is to go a step further: explicitly train an LLM (using RL) to develop a new pseudo-language for compression – essentially teaching it to "think" or "speak" in a distilled shorthand that other models can decode.

Method: Training an LLM to Compress via RL

How can we practically train a model to generate this meta-language pseudocode? We envision a two-stage process:

1. Supervised Bootstrapping with Handcrafted Examples. We first create a dataset of input-output pairs where the output is a manually written compressed pseudocode for the input text. These might be handcrafted examples illustrating how to represent various kinds of information in a concise, structured form. For instance, a paragraph explaining a recipe could be paired with a terse code-like list of ingredients and steps (using abbreviations and numeric tokens), or a long clinical note paired with a pseudo-medical code summary. During this stage, the LLM is fine-tuned (or instruction-tuned) to map inputs to the target pseudocode, learning the basic format and vocabulary of the compression language. Think of this like teaching a human the syntax of a new shorthand: we show enough examples until the model gets the pattern.

2. Reinforcement Learning with AI Feedback. Supervised learning can teach the model the rough idea, but to truly optimize compression we employ reinforcement learning (RL). In this loop, the LLM (the policy we are training) will produce compressed pseudocode for a given input, and we will measure a reward for how good that output is. Designing the reward is crucial – it should encourage two things: (a) Fidelity, i.e. the original content can be perfectly reconstructed from the pseudocode, and (b) Brevity, i.e. the pseudocode is as short as possible (uses significantly fewer tokens than the original). Achieving both is a balancing act, since the model might be tempted to compress aggressively at the cost of losing meaning, or preserve everything at the cost of long output. We can combine these criteria into a single reward score. For example, we could use a large, pre-trained LLM as an evaluator: give it the pseudocode and ask it to reconstruct or interpret it, then compare the reconstruction to the original input. If the reconstruction is semantically close (e.g. high similarity or exact match on key facts), reward is given; if details are missing or altered, penalty is applied. This idea is analogous to "reinforcement learning from AI feedback", where a stronger model judges the weaker model's output . Notably, Jung & Kim (2024) followed a similar RL strategy for prompt compression: they evaluated fidelity by comparing the responses of an LLM given the original prompt vs. the compressed prompt, and rewarded the policy when the outputs were similar . In our case, the evaluator could be the same

LLM in a separate decoding run or a more powerful model (like using GPT-4 to judge a smaller model's pseudocode). The reward would also heavily favor shorter outputs (e.g. a bonus for each token eliminated or a target compression ratio). Using policy gradient methods (like PPO or REINFORCE), the model can then be tuned to maximize this reward.

Crucially, the output remains a sequence of standard tokens, not a binary blob. This means the compressed code might look like a mish-mash of words, symbols, and abbreviations – but all drawn from the vocabulary the model already knows (across all languages it was trained on). Past approaches to prompt compression often relied on training special embedding vectors or token indices that are not human-readable . However, those "soft prompts" are opaque and usually tied to one model – you can't take a learned embedding from GPT-3 and use it in PaLM, for example, and you can't interpret what it means . Here we insist on discrete, interpretable tokens. The pseudocode may not be immediately readable English, but developers or domain experts could learn its notation, and more importantly any LLM that knows the underlying vocabulary can potentially decode it. This approach is similar in spirit to manual prompt engineering: we're just automating and optimizing it with RL. It's also akin to an autoencoder, except implemented within the language space – the model "encodes" text into a shorter text. In fact, researchers have recently proposed adding an in-context autoencoder module to LLMs for compressing long contexts . Our approach differs in that we don't add new modules, we train the model's natural language output to become an effective code. That said, in the future one could combine these ideas – e.g. use a specialized decoder model to decompress the pseudocode, allowing even more aggressive compression. For now, we focus on the scenario where no custom decoder is needed, only standard LLM capabilities.

During RL training, we might iterate like this: feed the model an input, get pseudocode, have the evaluator model attempt to reconstruct or at least grade the pseudocode, and give a reward. Over many examples, the compressor model should improve at capturing all meaning in minimal tokens. We likely also use curriculum learning: start with shorter texts and high-level summaries, then gradually move to longer and more complex inputs as the model's pseudo-language evolves. By the end, the model ideally has converged on a compact "language" or notation for encoding information, shaped by the reward to be as information-dense as possible while still decodable.

Illustrative Examples of Semantic Pseudocode

What does this compressed meta-language look like in practice? It will resemble a fusion of natural language, code, and shorthand. The exact style would emerge from training, but we can provide a couple of illustrative examples to make it concrete. One example comes from a cooking recipe domain, and another from a medical domain. These are formatted as code blocks to reflect their structured nature.

Example 1: Compressing a Recipe Instruction

Let's say we have a recipe written in full prose. The model might compress it into a pseudocode listing ingredients and steps succinctly. For instance:

```
§CKP1§

I[
    Cf500,O1,G1,E1,Cr2s,S&P,Bt30;     # 肉饼 4st
    Pt600,Mlk100,Bt50;                  # purée
    Rd80,Cc80,Mx*,OO2s,Lz+              # insalata
]
P(
    1: min Cf+O+G+E+Cr+S&P → 8p • seal Bt • fry 3+3 • oven 180°×10;
    2: boil Pt 15 → mash + Mlk + Bt + S;
    3: slice Rd+Cc+Mx → dress OO + Lz + S;
    4: plate K | P | I
)
**« Entpacke TXT in lit. Deutsch» »**
```

What are we seeing? This pseudocode uses a structured format to capture a recipe for, say, cottage pie (meat pie with potato purée and a salad). The section I[...] lists Ingredients with quantities: e.g. Cf500 might mean 500g of ground beef (carne de Cf possibly stands for some ingredient, "**肉饼**" is a Chinese note for meat pie filling), O1,G1,E1 could be 1 onion, 1 clove of garlic, 1 egg, Cr2s might mean 2 slices of carrot, S&P is salt & pepper, Bt30 is 30g of butter. The second line has Pt600 (600g potatoes), Mlk100 (100ml milk), Bt50 (50g butter) for the purée; the third line looks like Rd80,Cc80 (80g radish, 80g cucumber), Mx* (mixed greens, quantity unspecified), OO2s (2 tbsp olive oil), Lz+ (perhaps "lemon zest, a bit"). These are followed by # comments in various languages (Chinese "**肉饼**", English shorthand "pur" for purée, Italian "ins" short for insalata maybe), showing that tokens from any language can be repurposed as long as the model and its tokenizer know them – a truly mixed meta-language.

The P(...) section enumerates Procedure steps. For example, step 1: "min Cf+O+G+E+Cr+S&P → 8p" likely means mince (min) all those ingredients and form 8 patties, then "seal Bt" (seal with butter), then "fry 3+3" (fry 3 minutes on each side), then "oven 180°×10" (bake at 180°C for 10 minutes). Step 2: boil potatoes 15 minutes, mash and add milk, butter, salt. Step 3: slice radish, carrot, mix with greens, dress with olive oil, lemon zest, salt. Step 4: plate everything (K | P | I might mean arrange Ketchup, Pie, Insalata on plate, or some plating code). Finally, the line «Entpacke TXT in lit. Deutsch» is a user instruction in Deutsch: "unpack the text in literary Deutsch In other words, the pseudocode above can be given to a Russian GPT instance which will then produce a full natural language recipe in Russian. The pseudocode maintained all the critical information (ingredients, amounts, actions, times) but in 1/3 or less the tokens of the original written recipe. A Deutsch (or any language) model can reconstruct it because it's composed of familiar symbols and words (albeit in compressed form). This demonstrates the language-agnostic nature of the code – it's not English or Chinese or Deutsch specifically, but a semantic representation that any sufficiently trained model can interpret and expand upon.

Example 2: Medical Note Compression and Expansion

For a more complex example, consider a medical case report. Such text is typically long and detail-rich. Here we show how it might be compressed and then expanded back:

Original (excerpt from a clinical note, English):

Michael is a 58-year-old male with an 8-year history of type 2 diabetes mellitus, a 10-year history of hypertension, and stage 3 chronic kidney disease. He presents with 4 days of worsening shortness of breath and bilateral leg swelling. He also reports a dry cough and fatigue. On examination, he has bilateral pitting edema in the ankles and mild crackles at the lung bases. Vital signs: blood pressure 160/95 mmHg, heart rate 88/min, O_2 saturation 94%. Laboratory tests show an elevated B-type natriuretic peptide (BNP) and a mild rise in creatinine. A chest X-ray reveals mild pulmonary edema. **Assessment:** Likely acute decompensated heart failure (CHF exacerbation). **Plan:** Begin IV diuretics, tighten blood pressure control, and follow up on renal function.

Compressed Pseudocode:

```
Pt[M58, T2DM@8y, HTN@10y, CKD3]
CC[SOB↑4d, leg_edema, cough(dry), fatigue]
Exam[edema++ ankles, crackles+ bases]
Vitals[BP160/95, HR88, O2%94]
Labs[BNP ↑, Cr ↑ mild]
Imaging[CXR → pulm edema mild]
Dx[CHF exacerbation]
Plan[IV diuretics, stricter BP ctrl, monitor renal fx]
```

Reconstructed (English)

A 58-year-old male patient (Michael) with an 8-year history of type 2 diabetes, a 10-year history of hypertension, and stage 3 CKD presents after 4 days of worsening shortness of breath and swelling in his legs. He also has a dry cough and feels fatigued. On exam, there is pronounced pitting edema in the ankles and mild crackles at the lung bases. His blood pressure is 160/95 mmHg, heart rate 88, and oxygen saturation 94%. Lab tests show an elevated BNP and a slight increase in creatinine. A chest X-ray shows mild pulmonary edema. **Diagnosis:** This is likely an acute exacerbation of congestive heart failure. **Plan:** Start IV diuretics, tighten blood pressure control, and monitor renal function closely.

Let's unpack the pseudocode format here. It reads almost like a SOAP note in shorthand. Pt[M58, T2DM@8y, HTN@10y, CKD3] encodes the patient: male 58, Type 2 Diabetes Mellitus for 8 years, Hypertension for 10 years, Chronic Kidney Disease stage 3. The chief complaint and history: CC[SOB↑4d, leg_edema, cough(dry), fatigue] means shortness of breath worsening for 4 days, leg edema, dry cough, fatigue. Exam findings: Exam[edema++ ankles, crackles+ bases] conveys bilateral ankle edema (marked as "++" perhaps to denote severity) and positive crackles in lung bases. Vitals[…] lists key vitals, Labs[…] key lab abnormalities (BNP up, creatinine mildly up), Imaging[…] shows chest X-ray finding, Dx[…] gives the diagnosis, and Plan[…] outlines the plan.

Despite being extremely terse (the entire patient presentation compressed into ~30 tokens), nothing crucial is lost – a doctor or an LLM can expand this back into full sentences, as shown. The pseudocode draws on common medical abbreviations (e.g. "SOB" for shortness of breath, "BP" for blood pressure, "ctrl" for control, "fx" for function) combined with symbolic notations ("↑" for increased, "@8y" for "for 8 years"). All these tokens are part of Unicode or common training data, so an LLM can understand them in context. **Notably, the compression here preserved every key fact**: patient demographics, each condition and duration, each symptom and its timeline, exam findings with location and severity, numerical vitals, lab trends, imaging result, diagnosis and plan. This shows the promise of a well-trained semantic compressor: it behaves almost like a domain-specific coding system (similar to how doctors use acronyms and shorthand), but one that *the AI learned automatically* and can translate from/to natural language.

These examples are hypothetical but grounded in real patterns. Importantly, the **meta-language pseudocode could be different for different domains** – the model might learn to use more medical jargon in medical contexts, more math/logic symbols in technical contexts, etc., all still using the general vocabulary it knows. With RL-based training, it will invent tokens or abbreviations that maximize compression while the evaluator still understands them. For instance, it might find that using "↑" and "↓" for increase/decrease, or omitting vowels, or using first letters of words, etc., are good strategies – essentially rediscovering a stenographer's tricks or creating a new pidgin language for the AI's own use.

Results and Potential Applications

If successful, an RL-trained semantic compression model could unlock several exciting benefits:

- **Longer Context and Memory:** Compressed representations mean we can pack more knowledge into the same context window. This could help with long documents in **retrieval-augmented generation (RAG)**. Rather than feeding raw passages, a system could compress each passage into pseudocode, allowing far more chunks to be included for the LLM to reason over. The model would then decompress or interpret them as needed. Similarly, for **storage** of conversation history or user data, compressed text takes up less space (both in RAM for live models and disk for archived data).
- **Faster Inference and Lower Cost:** Fewer tokens to process means faster turnaround and less compute. OpenAI's GPT-4, for example, is slower and costlier with longer inputs – if a user query and context can be sent in 100 tokens instead of 500, that's a direct saving. One study on prompt pruning (Jung & Kim 2024) achieved ~25% token count reduction with minimal performance loss, simply by dropping low-importance tokens . Our approach could potentially compress even further by *rephrasing* rather than dropping – 50% or more reduction – without loss of information. This translates to real dollar savings and speedups, especially at scale. Companies could serve more users or run models on smaller hardware if each prompt/response is shorter.
- **Improved Domain Solutions:** A compression model can be **fine-tuned per domain** to capture that domain's specific semantics efficiently. In medicine, as shown, it might use medical shorthand; in legal, it might learn a stylized legal code for case facts; in programming,

it might condense code and documentation into a dense pseudo-code. Domain-specific finetuning can improve the compression fidelity since the model can leverage domain terminology and predictable patterns. Notably, Gilbert et al. found GPT-4 handled code compression particularly well, likely because it "knew" how to concisely describe code functionalities in a consistent way . Likewise, a finance-trained model might compress balance sheets or contracts far better than a generic model. This specialization can enhance applications like **education** (condensing textbooks into cheat sheets) or **law** (summarizing legal documents into key statutes and precedents) while preserving critical details.

- **Multi-Agent Communication:** In advanced AI systems with multiple agents or tools interacting (think of an AI assistant that queries another tool or coordinates with a robot), a compressed interlingua can be valuable. Agents could communicate in the pseudocode language to convey complex information quickly. For example, an AI planner could send a robot an instruction encoded in a very compact form that the robot's model can decode into a detailed plan. This could reduce bandwidth in communication-limited settings (like a robot receiving updates over a network with latency) and enforce a kind of structured information passing. In effect, the agents develop an *efficient private language* for their dialogues. This idea resonates with research in multi-agent reinforcement learning where agents invent communication protocols to maximize team performance. Here, however, the "protocol" is explicitly about compressing semantics.
- **Training Smaller/Faster Models:** We can even turn this around – instead of always decompressing to full language for a smaller model to handle, we could try to train smaller models to **directly work in the compressed space**. For instance, a 6B-parameter model might struggle to write a long essay given a long prompt, but if the prompt is compressed pseudocode, maybe the smaller model can learn to reason over that concise representation (which contains the distilled facts without the distracting fluff). This is speculative, but it hints at a form of knowledge distillation: the large model compresses knowledge and the small model is trained to use that compressed knowledge to perform tasks. Over time, the smaller model might effectively "think" in the pseudo-language, benefiting from the semantic density (somewhat like how humans use note-taking and mental abstractions to cope with complexity). There is early discussion in AI circles that once you have a powerful compressor/decompressor, you can generate a lot of paired data (text ↔️ pseudocode) and train other models on it to align their reasoning with this more efficient representation. In essence, the **compression language could become a common lingo** that even smaller models understand, making them more capable without massive scale.
- **Robotics and VLA (Vision-Language-Action) Plans:** For **Vision-Language-Action** models (robotics or embodied AI that sees, communicates, and acts), concise instructions are gold. These systems often need to interpret high-level goals and translate to low-level actions under resource constraints. If a vision system describes a scene or a task in 10,000 words, a robot's action planner might bog down; but if that description is compressed into a 100-token plan code, the core information is there without overloading the planner. Moreover, a structured pseudo-language could be designed to interface well with planning algorithms (e.g. containing key objects, relationships, and steps in a format the action model can easily map to its capabilities). As an analogy, PDDL (Planning Domain Definition Language) in classical AI planning succinctly describes goals and states for planners – here the pseudocode could play a similar role for learned planners. Overall, compact representations could make real-time decision-making in complex environments more feasible.

In summary, a successful semantic compressor unlocks **efficiency gains at every stage**: prompting, inference, multi-step reasoning, storing knowledge, and transferring skills between models. It strives to overcome the "compression tax" that current LLMs implicitly pay – today's models compress internally

in ways we can't control, sometimes losing nuance , but by explicitly training compression we aim to preserve meaning *and* reduce cost.

Future Directions and Considerations

While this vision is promising, there are important challenges and open questions. Here are a few future directions and **safety/ethical considerations** to keep in mind:

- **Specialized Decoders and Architecture Improvements:** Our approach kept outputs decoder-agnostic (usable by any LLM). But one could achieve higher compression by introducing specialized decoders or architectures. For example, a two-part system where one model compresses into a very tight code (perhaps even binary or pseudo-binary tokens) and a small *custom* model solely trained to decompress it. This starts to resemble classic compression algorithms (encoder/decoder pairs). Some research already explores adding dedicated compression modules to LLMs (e.g. the In-Context Autoencoder that adds 1% parameters to achieve 4× compression on long texts) . The trade-off is complexity and generality: if each model requires a custom decoder, we lose the plug-and-play appeal. Future work may hybridize these ideas – for instance, using a general pseudocode for most data but a very compact code for certain very structured data, with fallback to a decoder.
- **Cross-Domain Consistency vs. Custom Jargon:** As mentioned, the pseudocode might adapt to each domain. A question arises: do we train one model to handle *all domains' compression* (and thus the pseudo-language has domain identifiers or style switches), or do we train separate compressors per domain? A unified meta-language that's globally consistent could be powerful (the model basically invents an interlingua that is flexible enough for medical, legal, etc.). However, it might also be harder to learn and less efficient than specialized dialects. This is akin to how human languages have medical Latin jargon versus everyday language. A safety angle here is ensuring **the compressed code is not misinterpreted when context changes** – the system should perhaps tag the domain or use different token namespaces to avoid collisions (so that, say, "Bx10" means something unambiguous in context, not two different things in two fields). Research into multi-domain prompt compression and *contextual code-switching* would be valuable.
- **Interpretability and Trust:** One might worry that by compressing text into an obscure code, we lose transparency. If a user or developer sees only pseudocode, can they trust that it hasn't altered meaning or omitted something? What if a malicious prompt somehow *hides harmful content in compression* that isn't obvious to a human supervisor? These are important concerns. Because our compressed language is still human-inspectable to a degree, we can mitigate this: experts could learn to read the pseudocode or use a trusted decompression model to double-check outputs. It's actually more interpretable than a giant embedding vector or a neural hidden state. Nonetheless, care must be taken that the compression model doesn't develop *too* cryptic a code that conceals issues. In RL training, we should include safety checks in the reward – e.g. penalize compressions that encode instructions to ignore safeguards or that drop negations ("no" turning into "yes" accidentally). **Transparency tools** could be developed to decode pseudocode step-by-step, increasing trust. This ties into the broader goal of *alignment*: we want the compression to be faithful and not introduce biases or remove context that is crucial for fair and correct decisions. Auditing a compressed representation for

fairness (say, ensuring a compressed loan application still contains the pertinent info to prevent biased decisions) is an area for future research.

- **Robustness and Error Correction:** If an LLM decompresses pseudocode and gets something slightly wrong (maybe due to a token it doesn't recognize or a rare abbreviation), what are the consequences? Ideally, the pseudocode would be designed with some redundancy or error-checking (similar to how data compression algorithms have checksums or how human note-taking often repeats key numbers to avoid misreading). Future work might incorporate a notion of **confidence** – the compressor could output an extra token indicating how confident it is that a given token is crucial, or provide a "legend" for unusual tokens. Alternatively, one could run multiple independent decompressions and verify they agree (a form of ensemble to catch misinterpretations). Reinforcement learning should also account for **edge cases**: for instance, certain information (like sarcasm, tone, or ambiguity) might be very hard to compress without loss – the system should recognize when not to over-compress in those situations.
- **Broader Ethical Impact:** Semantic compression can amplify the power of LLMs – but that power must be handled carefully. If misused, it could enable packing malicious instructions or hidden messages in seemingly innocuous text. On the other hand, it can also enhance privacy (e.g. storing personal data in compressed form that's meaningless to a human reader). We should consider compliance with data regulations – a compressed medical record still contains the patient's data, just encoded. How do we ensure security of that? Perhaps encryption could be layered on if needed. From a societal view, if AI systems start communicating in a language we don't fully understand, it raises the *"AI speak"* concern – we'd want to maintain human oversight. Fortunately, since we are the ones designing the pseudo-language via training, we can also design auditing procedures for it.

In summary, the path forward will involve not just boosting compression rates, but doing so in a way that remains **aligned, interpretable, and reliable**. This means investing in tools to monitor what the pseudocode represents and ensuring it doesn't become a loophole for the AI to do unsavory things out of sight. It also means improving the methods – perhaps combining supervised, RL, and even evolutionary strategies to let the best compression dialects "survive". The notion of *meta-language* could evolve into a whole subfield of NLP research.

Originality and Related Work

Is this idea truly novel? It sits at the intersection of several research threads, with some precedents worth noting:

- **Prompt Compression and Token Pruning:** Over the last year or two, a number of works have looked at compressing prompts to address context length limits. For example, *Selective-Context* (Li et al., 2023) uses a smaller model to identify and drop least-informative sentences in a prompt . **LLMLingua** (Jiang et al., 2023) introduced a multi-step compression (with a "budget controller" and iterative token removal) that achieved up to 20× compression with minimal loss by carefully preserving key tokens . These approaches, however, mostly rely on *extractive* compression – they keep or cut tokens from the original text. Our pseudocode idea is more **abstractive**, generating new tokens (abbreviations or symbols) that were not in the original, to pack meaning more densely. Abstractive summarization has of course long been

studied in NLP, but summarization usually prioritizes readability and often loses granular details, whereas our goal is a *faithful, if not human-ready, re-encoding*.

- **RL-Based Compression Methods:** There have been a few recent works applying reinforcement learning to prompt compression. Jung & Kim (2023) proposed **PCRL (Prompt Compression with Reinforcement Learning)**, which trained a policy network to **edit prompts** by removing tokens, optimizing a reward that balanced prompt length against the quality of the model's answer with that prompt . They managed ~25% token count reduction on average without degrading performance . Another approach, **TACO-RL (Task Aware Compression with RL)** by Shandilya et al. (2024), fine-tuned an encoder model with on-policy RL to compress prompts in a *task-specific* way, using GPT-4 outputs as guidance for what information is crucial . TACO-RL explicitly considers the task's reward (e.g. whether a question is answered correctly using the compressed prompt) and improved task performance significantly over non-RL baselines . These works show that RL can successfully tune models to drop or keep tokens for compression. However, they generally stop short of inventing new notation – they're pruning or slightly paraphrasing natural language. An exception is a reported experiment by Chuang et al. (2024) where a generative model was trained with RL to compress prompts (not just prune) . It had high overhead and was tested on classification tasks. This comes closest to the idea of an RL-trained abstractive compressor, though possibly without the focus on a persistent pseudocode language.
- **Semantic Compression Research:** Gilbert et al.'s study "Semantic Compression with Large Language Models" (2023) is a direct precursor, demonstrating that GPT-3.5 and GPT-4 can compress and decompress text and code when prompted appropriately . They evaluated things like compression ratio and reconstruction quality, comparing to standard algorithms . The key difference is they did not train new models or use RL – they essentially tested prompting capabilities. The results were promising (GPT-4 especially was able to preserve meaning at high compression), but also highlighted limitations (GPT-3.5 struggled in some cases, and there was no guarantee of consistency in the "language" used for compression) . Our approach can be seen as taking that insight ("LLMs can compress themselves") and formalizing it with a training procedure to enforce consistency and optimality. By training a model specifically to produce a *pseudocode-like compression*, we avoid problems like the model randomly choosing different compression styles each time. It standardizes the meta-language.
- **Autoencoders and Learned Representations:** In the broader machine learning context, *autoencoders* and *representation learning* are all about compressing data into a lower-dimensional form. Classic autoencoders, however, compress into a vector (latent variables) not meant to be read by humans. There is recent work like **In-Context Autoencoder (ICAE)** by Ge et al. (2024) that adds an autoencoding capability to LLMs, compressing long text into a small "memory slot" within the context . They achieved about 4× compression on average with a slight model augmentation. This is somewhat orthogonal to our idea, because they modify the model architecture and still rely on the model's internal state to carry compressed info, whereas we keep everything in the output token stream. The advantage of our approach is you can transfer the compressed text between different models or store it externally. Autoencoder-based solutions might achieve higher compression ratios, but they function more like an internal memory for a single model.
- **Emergent Languages in Multi-Agent Systems:** Although not directly about LLM prompt compression, it's worth noting the parallel in multi-agent reinforcement learning where agents develop **emergent communication protocols**. In those setups, two or more agents invent a symbolic language to coordinate on tasks, often under pressure to make it efficient. They tend to develop short codes for important concepts (much like humans develop slang) . Our single-model setup is analogous in that the compressor and decompressor (which could be the same model or two instances) form a sender-receiver pair. In fact, one could frame our training as a **self-play game**: one instance compresses, another decompresses, and the reward is high if the final output matches the original input. This is essentially how one might train it with RL (it

becomes like two agents communicating). Such emergent language research suggests that the **bottleneck** (limiting message length) combined with a *need for accuracy* will naturally create a compressed code. The twist here is we initialize it with human-understandable tokens and push it further with RL and possibly larger models' guidance.

- **Patents or Specific Implementations:** As of this writing (mid-2025), we are not aware of any patents that specifically cover "RL-trained LLM for pseudocode compression." There are patents on using RL to compress machine learning models themselves (like compressing weights or architectures), which is a different domain (model compression vs. prompt/data compression) . The idea of using a pseudocode-like intermediate for NLP tasks is somewhat novel, though there have been related ideas such as using pseudocode representations for solving math problems or code generation (where a model generates pseudocode before actual code to improve reasoning). However, those are not about compression.

Given the above, the concept of a **reinforcement-learned semantic compression language** appears to be **novel in its specific formulation**, though it certainly overlaps with the trend of prompt optimization. Our approach distinguishes itself by emphasizing a *structured, reusable language* for compression, rather than one-off prompt hacks or opaque embeddings. It takes inspiration from many areas (summarization, knowledge distillation, multi-agent communication, autoencoders) and combines them. In doing so, it opens up questions like: *How compact can such a language get before models start losing nuance? Will different initializations converge to similar compression codes, or arbitrarily different ones? Can humans find these representations useful (e.g., as a new form of notation)?* The literature so far provides glimpses that LLMs are capable of this kind of semantic compression and that RL can fine-tune sequence outputs effectively . So, we are standing on a foundation of existing research, but pushing into new territory by proposing a deliberate *language creation* process via RL for the purpose of extreme token efficiency.

Conclusion

In an era where **context is king** and every token counts, training LLMs to develop a compressed pseudocode could be a game-changer. It's a strategy to make AI communication more efficient without sacrificing the richness of information – essentially finding a better trade-off on the compression vs. fidelity curve. We discussed how this could work: seeding the model with examples, refining with reinforcement learning using larger models as judges, and ensuring the compressed code remains human-auditable and model-agnostic. The examples in recipes and medicine show that even complex instructions and data can be squashed into a fraction of their length if done cleverly.

This idea could amplify the capabilities of LLMs: allowing them to ingest longer texts, remember more, work faster, and even collaborate more effectively by sharing a private language. It aligns with the insight that *compression is understanding* – if an AI truly understands, it should express that understanding efficiently . We also caution that with great compression comes great responsibility: we must keep the process interpretable and aligned with human intentions to avoid creating a gibberish code that we can't oversee.

The concept of an AI meta-language for knowledge representation is an exciting frontier. As research progresses, we'll learn whether an RL-trained pseudocode is just an interesting experiment or a practical component of next-generation AI systems. Given the rapid progress in both large models and training techniques, don't be surprised if your future AI assistant communicates with its peers in what looks like an alien cipher – it might just be the optimal way to save time while saying everything that needs to be said.

Sources:

1. Gilbert et al., *"Semantic Compression with Large Language Models,"* arXiv preprint, 2023 – showing GPT-4 can compress prompts and later reconstruct them, preserving intent .
2. Jiang et al., *"LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models,"* arXiv 2023 – introducing prompt compression with budget control, achieving up to 20× compression with minimal performance loss .
3. Jung & Kim, *"Discrete Prompt Compression with Reinforcement Learning (PCRL),"* arXiv 2024 – using RL to train a policy that prunes tokens, yielding ~25% shorter prompts while retaining task performance .
4. Shandilya et al., *"TACO-RL: Task Aware Prompt Compression Optimization with Reinforcement Learning,"* arXiv 2024 – combining supervised and RL (with GPT-4 feedback) to compress prompts in a task-specific way, improving downstream task accuracy significantly .
5. TechXplore (Ingrid Fadelli), *"Algorithm based on LLMs doubles lossless data compression rates,"* 2025 – summary of Li et al.'s work in Nature Mach. Intell., emphasizing that better understanding enables better compression (e.g. LMCompress algorithm) .
6. Masood, *"Beyond the Benchmarks…,"* Medium, 2025 – discusses the compression bias in LLMs' internal representations, noting that they compress aggressively which can lose nuance , highlighting the need for more meaning-preserving approaches.
7. Ge et al., *"In-context Autoencoder for Context Compression in an LLM,"* ICLR 2024 – proposes adding an autoencoding module to LLMs to compress long contexts ~4×, indicating the interest in solving long-context issues via learned compression .
8. TACO-RL paper (excerpt of related work), 2024 – categorizes prompt compression methods (token pruning, abstractive, soft prompts) and notes prior RL approaches for prompt compression .
9. DataCamp Blog, *"RLAIF: Reinforcement Learning from AI Feedback,"* 2024 – explains using AI (like LLM-as-a-judge) to provide reward signals instead of humans , a concept applicable in our RL setup for compression.
10. Jung & Kim (PCRL) paper, Table 1 and text, 2024 – highlights limitations of soft prompt (embedding) compression (not interpretable, fixed length, not cross-model) and the benefits of discrete token compression , aligning with our decision to use pseudocode tokens for generality.