Aaron Script:

Good afternoon. Today, we will be presenting our project on Dual Compression for Agentic AI: Optimizing Models and Prompts in LLMs.

So first, I will briefly introduce Agentic AI. These are autonomous systems that can make decisions and set goals proactively, adapt their behavior to dynamic environments, and combine large language models with memory and planning functions, which makes them more than just responsive tools.

Now, to present the solution architecture proposed in the initial paper. Our approach transforms unstructured prompts into well-defined combinatorial search problems through what we call Search Space Specification.

The Pattern Library provides composable prompt patterns as a domain-specific language that allows us to create structured reasoning graphs. At the core, we implement a Successive Halving Optimizer that uses efficient bandit optimization to avoid brute-force grid search.

This architecture allows us to take any input task and deliver optimized agent programs specifically tailored to particular task distributions. The key innovation here is enabling unified optimization across diverse prompt engineering challenges.

The paper's evaluation shows some critical findings. The red boxes in the table demonstrate significant performance gaps in compressed models, clearly indicating why model compression alone is not sufficient.

However, the blue boxes reveal how specialized prompting can recover performance in these compressed models, which points to the necessity for prompt compression as well.

Our solution is a dual-compression approach that jointly optimizes both model architecture and prompt structure. This approach dynamically balances compression ratios based on task requirements and implements efficient bandit optimization to avoid brute-force grid search.

Lucas Script:

What is model Compression:
Not scalable or fast: The current LLMs are Huge and require full model fine tuning for new tasks.
Inefficient for Multi-Tasking jobs: **Multi-tasking LLMs** suffer from *prompt interference*, *memory overload*, and *data drift*.
But here's the catch — when you compress too much, you lose important context, and

task performance drops. This is where naive compression falls short

These methods essentially trade off model quality in return for reduced latency and model size. Thus, there is an
inevitable trade-off between accuracy and efficiency, resulting in a noticeable reduction in the model's
accuracy and, consequently, the overall performance benefits of LLMs

With the figure 2 showing, Both the Quantization method and Pruning Method remain closer to that of the full LLaMA model when we prune 50% or quantize the parameters to 4 bits,. But later the perplexity consistently increases as we decrease the allocated resource.

On the left, we have responses from the original, full LLaMA-7B model — accurate and complete answers to basic questions.
But after we prune 62.5% of the model's weights — as shown in the second column — the responses become **unrelated or off-topic**, even for simple factual prompts.

Now in the third column, we add a **manually designed hard prompt**. This helps the model recover **some** useful output — but it's still inconsistent.

The real breakthrough is in the fourth column:
Using a **learned soft prompt (adaptive prompt)**, the compressed model not only improves response quality, but recovers **accurate, transferable answers** across all three questions — *without retraining the model itself*.

Formulation:

"Here, we formulate how to optimize prompts specifically for compressed LLMs.
The goal is to train a prompt that can recover the performance of a compressed model — like a quantized or pruned LLaMA — without fine-tuning the model itself."

"We start with the core idea — only the prompt is learnable.
Given a compressed model with frozen parameters {theta}θ~, we prepend a trainable sequence of soft prompt tokens to each input and minimize the negative log-likelihood of the next token."
"This lets the prompt adapt to the compression artifacts of the model, even though the model weights themselves remain fixed."

"For training, we use a standard text dataset like C4.
The model is already compressed — through pruning or quantization — and we use an optimizer like AdamW to update only the prompt embeddings.
It's worth noting that the training process is lightweight, since we're not touching the full model."

"To ensure practical deployment, we limit the number of soft tokens to a maximum length, denoted (tau)τmax, such as 20.
Also, the model weights {theta}θ~ are completely frozen — this keeps the method fast and avoids overfitting.
The learned prompt is shared across all inputs."

"One of the most surprising results is how well these prompts generalize.
Prompts trained on C4 transfer well to other datasets like Wikitext-2 and PTB.
They also generalize across compression types and levels — for example, a prompt trained on a 4-bit quantized model still helps on a 2-bit model.
And even more impressively, the same prompt improves performance on completely different tasks, like question answering — all in zero-shot."

Experimental result:
We can observe there is still a significant improvement with 25 prompt tokens, and we can improve the performance by increasing prompt size.

The figure 3 at the right side validates the impact of this approach on the set of C4. We can observe a significant improvement in PPL across all compression levels.
And by employing soft prompt tokens, the compressed LLMs using SparseGPT with 50% sparsity even out-perform the full model counterpart. This trends also observed in
The 4 bit quantization of LLM using GPTQ.

Further, even with the further enhanced compression, the compressed LLMs with soft prompt tokens learned from the Equation still maintain comparable PPL to their original counterparts.

Benjamin Script

1. What is DCP?

Dynamic Compressing Prompts (DCP) is a tiny “agent” that reads a long prompt token-by-token and decides, for each token, whether to **keep** or **drop** it.

At each step the current prompt becomes the **state** s_t, the agent outputs an **action** a_t (a keep/drop mask), and the masked prompt forms the **next state** s_{t+1}.

A reward is computed by Equation 4 and fed back to update the agent.

Training follows a Hierarchical Prompt Compression (HPC), so the agent first learns easy and light compression and then graduates to aggressive pruning via Proximal Policy Optimization (PPO).

After a few such loops the final, much shorter sequence is decoded, giving an **output prompt** that preserves meaning but costs fewer tokens.

2. Core Optimization Formulation

Objective (Eq. 1).

Equation 1 gives the objective. First term (KL-divergence) – keep the *output distribution* produced from the compressed prompt \tilde{x} as close as possible to the distribution you would have obtained with the full prompt x.
Second term (\rho) – the fraction of tokens that survive compression. Smaller \rho ⇒ greater savings.
DCP is trying to *balance* two goals: aggressive length reduction and minimal loss of model behaviour. Equation (1) formalises that trade-off.

Reward function (Eq. 4).

Equation 4 is the reward function of reinforcement learning.

alpha/rho — reward for making the prompt shorter.
D (BERTScore) — reward for retaining key information.
KL — penalty when the LLM's answers drift from those produced by the original prompt.

This reward lets the agent learn *without querying the expensive target LLM during training* yet still balances length, fidelity and information retention.

PPO Update (Eq. 7)

Equation 7 defines the PPO update.

delta is the probability **ratio** between new and old policies, while A_old is the **advantage** telling how much better or worse that action was than average.

The expression min(delta A_old, clip A_old) chooses the clipped version whenever the ratio would push the policy outside a trust region of width epsilon.

Maximising J with mini-batch (stochastic) gradient ascent therefore *increases* the probability of advantageous actions but never by more than the safe clip factor.

In short, the PPO update: **reward good moves, punish bad ones, and cap every step for stability**.

3. Key Results

TABLE I
PERFORMANCE OF DIFFERENT METHODS ON THE CONVERSATION (SHAREGPT) AND SUMMARIZATION (ARXIV-MARCH23) TASKS.

Method	Pub.'Year	BLEU ↑	BLEURT ↑	Rouge-1 ↑	Rouge-2 ↑	Rouge-L ↑	BS F1 ↑	Tokens ↓	$1/\rho$ ↑
ShareGPT									
Selective-Context [20]	EMNLP'2023	38.53	-0.21	51.27	38.35	43.51	78.30	183	3.3x
LLMLingua[18]	EMNLP'2023	38.71	-0.21	51.43	38.62	43.57	78.27	186	3.2x
LLMLingua-2-small [12]	ACL'2024	56.79	0.37	76.09	58.47	63.56	89.54	191	3.1x
LLMLingua-2 [12]	ACL'2024	61.97	0.47	78.64	63.07	67.50	90.87	184	3.3x
LLM-DCP (Ours)	-	**64.93**	**0.54**	**80.24**	**65.54**	**69.89**	**91.80**	**175**	**3.4x**
Arxiv-March23									
Selective-Context [20]	EMNLP'2023	8.83	-0.61	43.43	13.46	18.92	73.75	933	11.8x
LLMLingua[18]	EMNLP'2023	5.70	-0.74	32.29	8.78	15.17	69.60	1276	8.7x
LLMLingua-2-small [12]	ACL'2024	8.56	**-0.45**	45.52	15.47	21.09	75.49	1017	10.9x
LLMLingua-2 [12]	ACL'2024	**10.84**	-0.57	48.49	14.62	19.95	75.15	920	12.0x
LLM-DCP (Ours)	-	10.10	-0.55	**48.81**	**15.94**	**21.63**	**75.91**	**855**	**12.9x**

TABLE II
PERFORMANCE OF DIFFERENT METHODS ON THE REASONING (GSM8K), AND IN-CONTEXT LEARNING (BBH) TASKS.

Method	Pub.'Year	*1-shot constraint*			*half-shot constraint*		
		EM ↑	Tokens ↓	$1/\rho$ ↑	*EM* ↑	Tokens ↓	$1/\rho$ ↑
GSM8K							
Selective-Context [20]	EMNLP'2023	76.57	436	5.4x	76.15	182	13.0x
LLMLingua[18]	EMNLP'2023	76.72	462	5.1x	77.02	174	13.6x
LLMLingua-2-small [12]	ACL'2024	75.66	425	5.6x	76.80	151	15.7x
LLMLingua-2 [12]	ACL'2024	76.87	415	5.7x	76.80	**140**	**16.9x**
LLM-DCP (Ours)	-	**77.03**	**343**	**6.9x**	**77.03**	153	15.5x
BBH							
Selective-Context [20]	EMNLP'2023	82.81	278	2.8x	81.91	152	5.1x
LLMLingua[18]	EMNLP'2023	81.68	271	2.9x	**84.72**	162	4.8x
LLMLingua-2-small [12]	ACL'2024	82.73	274	2.8x	82.12	155	5.0x
LLMLingua-2 [12]	ACL'2024	82.41	255	3.0x	82.64	145	5.3x
LLM-DCP (Ours)	-	**83.16**	**251**	**3.1x**	83.98	**145**	**5.3x**

Across four different benchmarks—dialogue, summarization, arithmetic reasoning, and chain-of-thought ICL—DCP *beats or matches* the strongest baselines **while squeezing 3–16 times more tokens out** of every prompt.

Q&A:
Eq.7

$A_{\pi_{\theta_{\text{old}}}}(s_t, a_t)$ **— what it really is**

1. **Baseline idea.** Every state s_t has a *value* V_{\phi}(s_t): the average return you can expect from there if you keep following the **old** policy.

2. **Actual return.** When the agent took action a_t, it saw an (estimated) return G_t — e.g.\ a one-step reward r_t or a multi-step rollout.

3. **Advantage definition.**

   A_{\pi_{\theta_{\text{old}}}}(s_t,a_t)=G_t-V_{\phi}(s_t)

   If you use the one-step version (shown in the paper's code) that becomes

   A = r_t - V_{\phi}(s_t)

4. **Interpretation.**

- **Positive** A → that action beat the baseline and should become *more* likely.

- **Negative** A → it under-performed and should become *less* likely.

5. **Why "old" policy?** During one PPO update you keep the policy fixed while you **measure** these advantages, so the subscript "old" just reminds us they're computed under the probabilities and value estimates *before* the current gradient step.

Aaron's Script

Looking ahead, we've identified three key directions for future research. First, in Dual Compression Potential, we want to develop a unified framework that simultaneously optimizes compression ratios across both model and prompt dimensions.

Second, for Agentic AI Architecture, we aim to maintain agent autonomy while significantly reducing computational requirements, designing architectures that dynamically allocate compression based on task needs.

Finally, our Empirical Validation needs expansion. While our current evaluation shows theoretical promise, we need comprehensive validation across diverse task types and complexity levels. Our next steps include conducting large-scale experiments across various domains and model families to fully validate our approach.